# Real-time Software Telemetry Processing System (RT-STPS) Version 4.1

## General

The NASA Goddard Space Flight Center's (GSFC) Direct Readout Laboratory (DRL), Code 606.3 developed the Real-time Software Telemetry Processing System (RT-STPS) software for the National Polar-orbiting Operational Environmental Satellite System (NPOESS) Preparatory Project (NPP) In-Situ Ground System (NISGS) and the International Polar Orbiter Processing Package (IPOPP).

Users must agree to all terms and conditions in the Software Usage Agreement on the DRL Web Portal before downloading this software.

Software and documentation published on the DRL Web Portal may occasionally be updated or modified. The most current versions of DRL software are available at the DRL Web Portal:

http://www.directreadout.gsfc.nasa.gov

Questions relating to the contents or status of this software and its documentation should be addressed to the DRL via the Contact Us mechanism at the DRL Web Portal:

http://directreadout.gsfc.nasa.gov/index.cfm?section=contact%20usAlgorithm

## Software Description

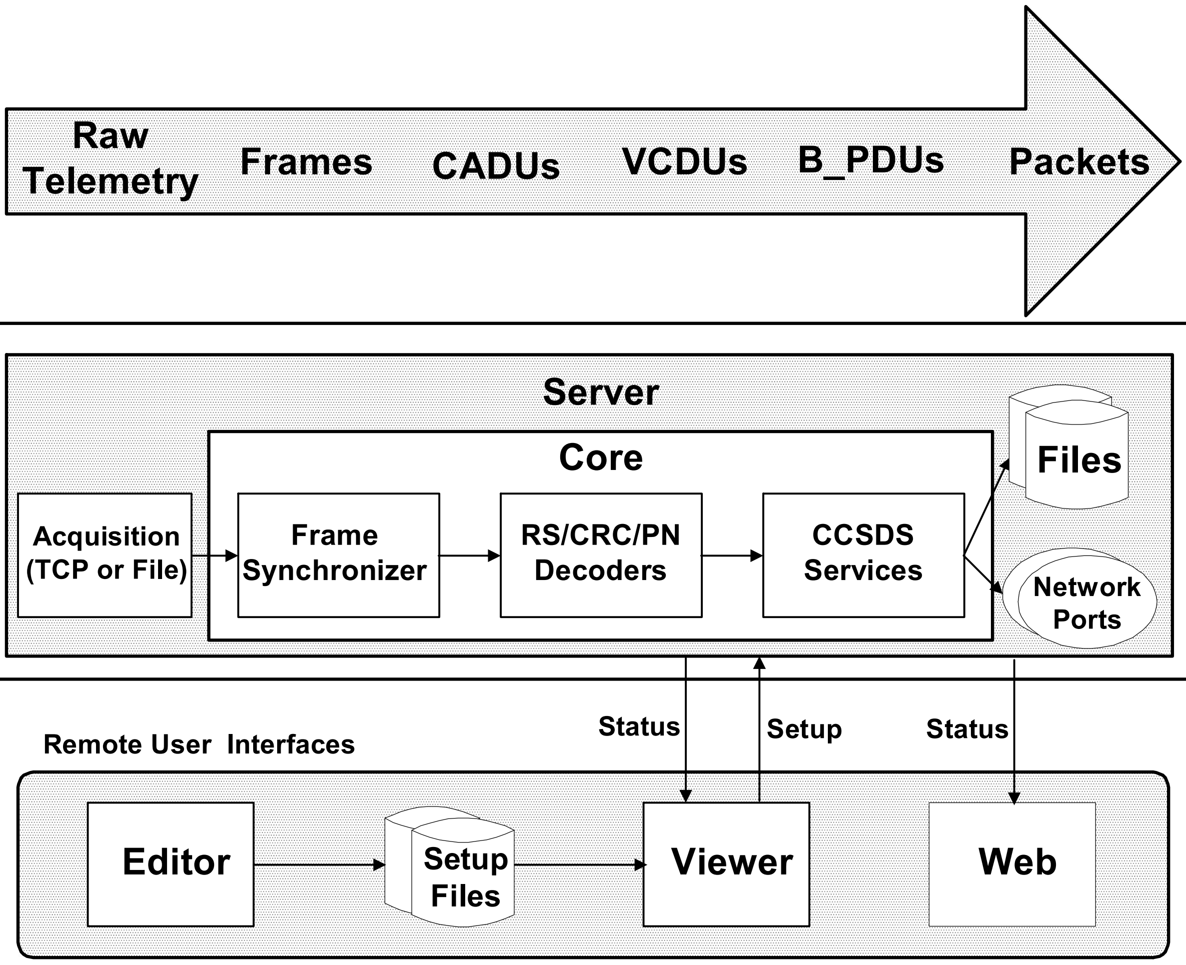
The RT-STPS Server and RT-STPS Batch Processor Java applications ingest raw telemetry data and produce Level 0 products, including sorted Consultative Committee for Space Data Systems (CCSDS) packets and Virtual Channel Data Units (VCDUs).

The RT-STPS Batch Processor ingests a single data file as specified in the configuration file. The RTSTPS Server runs continuously. The RT-STPS Server and Batch Processor can each send data across Transmission Control Protocol (TCP)/Internet Protocol (IP) ports, to files, or to both simultaneously, depending on the configuration file.

The RT-STPS package includes three utilities: the Viewer, the Server, and the Sender. The RT-STPS Viewer displays the progress of the RT-STPS Server as it runs, and loads and unloads the Server configuration files. The RT-STPS Sender copies a raw data file to the RT-STPS Server. The Rat is a rate buffer utility described further in Appendix C, "RT-STPS Tools."

The RT-STPS architecture is depicted in the following diagram. The components are further explained in Appendix A, "RT-STPS XML Configuration Files." For a more detailed description of RT-STPS and other DRL technologies, go to:

http://directreadout.gsfc.nasa.gov/index.cfm?section=technology



**RT-STPS Architecture**

## Software Version

This software package contains RT-STPS Version 4.1. Version 4.1 addresses minor issues associated with some of the arguments in the scripts files, and updates the version string to the proper release number. Version 4.1 has been tested with RS interleaves of 1, 4 and 5.

## Prerequisites

To run this package, you must have the Java Development Kit (JDK) or Java Runtime Engine (JRE) (Java 1.5 or higher) installed on your computer. If you plan to rebuild the RT-STPS distribution, you must install the JDK. Otherwise, only the Java Runtime Environment (JRE) must be installed. Ensure that Java applications can be executed.

## Program Inputs and Outputs

RT-STPS inputs are raw telemetry data and XML configuration files. (Configuration file examples are described in Appendix B.) RT-STPS outputs Level 0 products including sorted CCSDS packets and VCDUs.

## Installation and Configuration

RT-STPS can be installed on Linux or Microsoft Windows platforms. Installation instructions for each platform are provided in separate sections, "Linux Platform Installation" and "Windows Platform Installation." Operating instructions for both platforms are contained in the "Program Operation" section.

## Linux Platform Installation

**NOTE:** The bin directory of the JRE (or JDK) must be added to the beginning of the PATH environment variable so that the 'java' executable can be executed (and the 'javac' and 'javadoc' executables, if you plan to rebuild). Placing the bin directory at the beginning of the PATH environment variable prevents use of another Java distribution (such as the Java that ships with some operating systems).

Copy the RT-STPS distribution to the desired location. The "rt-stps-<version>.tar.gz" file (where <version> is the version number) contains, at the top level, a directory named "rt-stps". In the desired directory (typically "drl"), copy "rt-stps-<version>.tar.gz" and enter:

```
tar xzf rt-stps-<version>.tar.gz
```

This will create the "rt-stps" directory in that directory. All directory paths referred to in these instructions are relative to the "rt-stps" directory.

### Configure RT-STPS

Configure the RT-STPS Server by editing the Extensible Mark-up Language (XML) configuration files in the "./configs" directory. Instructions for editing the configuration files and the XML definitions are provided in Appendix A.

**NOTE:** If a previous version of RT-STPS was installed on your computer, we recommend that you update the configuration files so that they contain the default parameters for Version 4.1.

### Configure the Java Service Wrapper

The RT-STPS Server is run as a service using the Java Service Wrapper (JSW), which is configured in "./jsw/conf/rt-stps-server.conf". The lines defining "wrapper.java.additional.1", wrapper.java.additional.2", etc., specify the RT-STPS Server parameters as described in Appendix B. The default values should work in most cases. If the default values need to be modified, the "wrapper.java.additional.x" lines must start with "x" as "1" and increase with no gaps.

In the "./jsw/bin" directory, there should be a soft logical link from "wrapper" to "wrapper-32" for a 32-bit operating system and Java. If a 64-bit operating system and Java are being used, while in the "./jsw/bin" directory, delete "wrapper" and set the soft logical link using "ln -s wrapper-64 wrapper".

Likewise, in the "./jsw/lib" directory there should be a soft logical link from "libwrapper.so" to "libwrapper-32.so" (for a 32-bit operating system and Java). If a 64-bit operating system and Java are being used, while in the "./jsw/lib" directory delete "libwrapper.so" and set the soft logical link using "ln -s libwrapper-64.so libwrapper.so".

**NOTE:** "wrapper" is a symbolic link to either "wrapper-32" or "wrapper-64". They should be marked as executable, and the symbolic link created to the appropriate file, depending on your architecture. If you are trying to run rt-stps and get a "link" error, check that the link is set up properly and that the files are set to executable. For example:

```
[nisgs@nisfes bin]$ ls -l
total 212
-rwx--x--x  1 nisgs nisgs  14370 Sep  5 22:52 rt-stps-server.sh
lrwxrwxrwx  1 nisgs nisgs  10 Sep  5 23:43 wrapper -> wrapper-32
-rwx--x--x  1 nisgs nisgs  89171 Sep  5 22:52 wrapper-32
-rwx--x--x  1 nisgs nisgs 101769 Sep  5 22:52 wrapper-64
```

### Start the RT-STPS Server Automatically at Boot

The RT-STPS Server can be started at boot. In /etc/rc.local add:

```
su - <user> -c "<path>/jsw/bin/rt-stps-server.sh start"
```

where <user> is the user the RT-STPS Server will run as, and <path> is the absolute path of the "rt-stps" directory. Do not run the RT-STPS Server as "root".

### Start the RT-STPS Server Manually

The RT-STPS Server can be started, restarted, or stopped at any time from a shell by entering :

```
"<path>/jsw/bin/rt-stps-server.sh <cmd>"
```

where <path> is the absolute path of the "rt-stps" directory, and <cmd> is <start>, <restart>, or <stop>. This must be done as the same user that originally started the RT-STPS Server.

### Troubleshooting

Log files for the JSW are in "./jsw/logs". They should be examined if the RT-STPS Server does not appear to be running.

### Rebuilding

If you find it necessary to rebuild the RT-STPS distribution, use:

```
./build.sh
```

to compile the Java source files and create the "rt-stps.jar" file in the "lib" directory.

**NOTE:** The build.bat file calls the Java Remote Method Invocation Compiler (RMIC) "rmic.exe" program to build the remote method invocation portion of RT-STPS. The GNU version of RMIC cannot be used with the RT-STPS buildscript.

Ensure that the Java rmic.exe program is before the GNU RMIC program in the system PATH, so the build.bat file will execute properly. Alternatively, edit the build.bat file and hard code the absolute path to Java rmic.exe.

### Create Launchers

You may create desktop launchers to run the RT-STPS Viewer and RT-STPS Sender. The creation of launchers varies between the different types of Linux and their desktop environments. In all cases the command lines described in "Execute the RT-STPS Viewer" and "Execute the RT-STPS Sender" will apply.

### Execute the RT-STPS Viewer

Double-click the RT-STPS Viewer launcher. Or, if a launcher is not being used, while in the "rt-stps" directory, enter":

```
./bin/viewer.sh
```

The RT-STPS Viewer can then be used to view the progress of the RT-STPS Server as it runs, as well as to load and unload configuration files. Additional instructions explaining the use of the RT-STPS Viewer are contained in the "Program Operation" section of this document.

### Execute the RT-STPS Sender

Double-click the RT-STPS Sender launcher. Or, if a launcher is not being used, while in the "rt-stps" directory, enter:

```
./bin/sender.sh
```

The RT-STPS Sender can then be used to send a raw data file to the RT-STPS Server.

Additional instructions explaining the use of the RT-STPS Sender are contained in the "Program Operation" section of this document. Other ways to control the RT-STPS Server are described in Appendix B.

### Batch Command

A batch command lets you run RT-STPS as a standalone, one-time program. You run it with a configuration file and an input data file containing telemetry, and it produces whatever output is specified in the configuration. At the command line, type:

```
./bin/batch.sh <configurationFileName> <dataFileName>
```

## Windows Platform Installation

**NOTE:** The 'java' executable will have been installed in "C:\WINDOWS\system32" on XP systems. If a different 'java' executable is to be used, the "bin" directory of its JRE (or JDK) must be added to the beginning of the PATH environment variable (i.e., before "C:\WINDOWS\system32"). If you plan to rebuild the RT-STPS distribution, the "bin" directory of the JDK must be added to the beginning to the PATH environment variable so that the 'javac' and 'javadoc' executables can be executed.

Copy the RT-STPS distribution to the desired location.

The "rt-stps-<version>.zip" file (where <version> is the version number) contains, at the top level, a directory named "rt-stps". Extract/copy the "rt-stps" directory to the desired location (typically a directory named "drl" in "C:\Program Files"). All directory paths referred to in these instructions are relative to the "rt-stps" directory.

### Java Setup and Performance Notes

Many of the scripts (batch files) in the RT-STPS "bin" directory employ the "-server" option to increase performance, as the Sun Java Virtual Machine (JVM) Server is said to be somewhat faster than the "client" JVM. By default the JRE comes with the client JVM only, and to use the Server JVM, you must install the JDK. Once the JDK is installed, either copy the server directory from "jdk/jre/bin" to the "jre/bin" directory, or adjust the PATH under Windows to place the "jdk/bin" directory on the path instead of the jre.

If the "-server" option is used in a batch file and the Server JVM is not properly installed, Windows will issue an error message as follows:

```
Error: no 'server' JVM at "...\jreX.X.X\bin\server\jvm.dll"
```

If the JDK is installed, set the PATH so that the "jdk/bin" is employed instead of the "jre/bin" directory in the environment variables under Windows. For example:
JAVA_HOME C:\java\jdk1.5.0_03

And then:

```
PATH %JAVA_HOME%\bin
```

The 'java' executable will have been installed in "C:\WINDOWS\system32" on XP systems. If a different 'java' executable is to be used, the "bin" directory of its JRE (or JDK) must be added to the beginning of the PATH environment variable (i.e., before "C:\WINDOWS\system32"). If you plan to rebuild the RT-STPS distribution, the "bin" directory of the JDK must be added to the beginning to the PATH environment variable so that the 'javac' and 'javadoc' executables can be executed.

### Configure RT-STPS

Configure the RT-STPS Server by editing the Extensible Mark-up Language (XML) configuration files in the "./config" directory. The XML elements found in each file are defined in Appendix A.

### Configure the Java Service Wrapper

The RT-STPS Server is run as a Windows service using the Java Service Wrapper (JSW), which is configured in ".\jsw-Windows\conf\rt-stps-server.conf". The lines defining "wrapper.java.additional.1", "wrapper.java.additional.2", etc. are used to specify the RT-STPS Server parameters as described in "rt-stps.txt". The default values should work in most cases. If the default values need to be modified, the "wrapper.java.additional.x" lines must start with "x" as "1" and increase with no gaps.

### Install the RT-STPS Server as a Windows Service

In the ".\jsw-Windows\bin" directory, double-click "Install-NT.bat". This will install the RT-STPS Server to be started automatically when Windows boots. To modify how the RT-STPS Server is installed, use the "Services" windows found at "Control Panel > Administrative Tools > Services". It will now contain a service named "RT-STPS Server". Using the "Service" window, the RT-STPS Server can be started/stopped. Whether or not the RT-STPS Server automatically starts at boot can also be specified.
To uninstall the RT-STPS Server, first stop the RT-STPS Server and then double-click "Uninstall-NT.bat" in the ".\jsw-Windows\bin" directory.

### Troubleshooting

Log files for the Java Service Wrapper are in ".\jsw-Windows\logs". They should be examined if the RT-STPS Server does not appear to be running.

### Rebuilding

If you find it necessary to rebuild the RT-STPS distribution, use:

```
.\build.bat
```

to compile the Java source files and create the "rt-stps.jar" file in the "lib" directory.

**NOTE:** The build.bat file calls the Java Remote Method Invocation Compiler (RMIC) "rmic.exe" program to build the remote method invocation portion of RT-STPS. However, if Cygwin is installed on your system, the distribution may include a program of the same name. The Cygwin version of RMIC cannot be used with the RT-STPS buildscript.

Ensure that the Java rmic.exe program is before the Cygwin rmic program in the system PATH, so the build.bat file will execute properly. Alternatively, edit the build.bat file and hard code the absolute path to Java rmic.exe.

### Creating Shortcuts

You can create a shortcut to the RT-STPS Viewer as follows:

a) In the ".\bin" directory, right-click the "viewer.bat" file and select "Create Shortcut".

b) Move the shortcut to the desired location (e.g., the Desktop).

c) Rename the shortcut "RT-STPS Viewer".

d) Right-click the shortcut and select "Properties". Modify the "Start in:" field to be the absolute path of the "rt-stps" directory. Change the "Run:" selection to "Minimized".

You can create a shortcut to the RT-STPS Sender as follows:

a) In the ".\bin" directory, right-click the "viewer.bat" file and select "Create Shortcut".

b) Move the shortcut to the desired location (e.g., the Desktop).

c) Rename the shortcut "RT-STPS Viewer".

d) Right-click the shortcut and select "Properties". Modify the "Start in:" field to be the absolute path of the "rt-stps" directory. Change the "Run:" selection to "Minimized".

### Execute the RT-STPS Viewer

If you are using a shortcut, double-click the RT-STPS Viewer shortcut. Otherwise, while in the "rt-stps" directory, enter:

```
.\bin\viewer.bat
```

The RT-STPS Viewer can then be used to view the progress of the RT-STPS Server as it runs and to load and unload configuration files. Additional instructions explaining the use of the RT-STPS Viewer are contained in the "Program Operation" section of this document.

### Execute the RT-STPS Sender

If you are using a shortcut, double-click the RT-STPS Sender shortcut. Otherwise, while in the "rt-stps" directory, enter:

```
.\bin\sender.bat
```

The RT-STPS Sender can then be used to send a raw data file to the RT-STPS Server.

Additional instructions explaining the use of the RT-STPS Sender are contained in the "Program Operation" section of this document. Other ways to control the RT-STPS Server are described in Appendix B.

### Batch Command

A batch command lets you run RT-STPS as a standalone, one-time program. You run it with a configuration file and an input data file containing telemetry, and it produces whatever output is specified in the configuration. At the command line, type:

```
.\bin\batch.bat <configurationFileName> <dataFileName>
```

## Program Operation

**NOTE:** The "Program Operation" section includes instructions for both Linux and Windows platforms.

The Batch Processor and Server each require a setup file. Typically, you load a setup prior to each pass, although the Server does have an auto-load feature, which is explained later in this document. Setup files and some sample setup files are stored in the "config" directory.

The data directory is the default target for output files, typically "../data" (Linux) or "..\data" (Windows). A sample raw telemetry file named "terracotta.dat" resides in these directories. It contains one virtual channel and one Application ID (APID) of Terra MODIS data (apid 64).

Note that server.sh, viewer.sh, and sender.sh do not exit immediately and therefore should be started in background, or in separate terminal windows, when executing from the command line.

### RT-STPS Server

Before attempting to start the Server, ensure that "drl/rt-stps" is the current directory. To run a quick test, first start the Server by running:

```
Linux:    ./bin/server.sh
Windows:  .\bin\server.bat
```

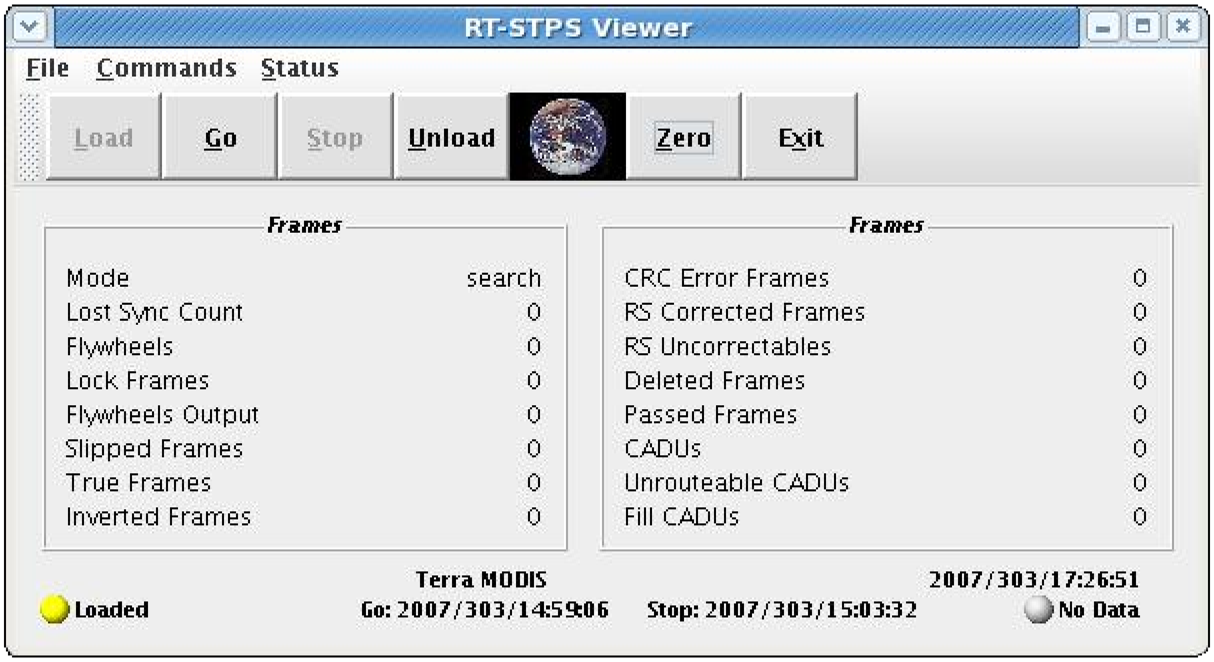
When ready, it prints "Ready to serve".

## RT-STPS Viewer

Next, run:

Linux:    ./bin/viewer.sh
Windows:  .\bin\viewer.bat

This opens the RT-STPS Viewer, depicted below, which can be used to view the progress of the RT-STPS Server as it runs, as well as to load and unload configuration files.



```
Frames                              Frames
Mode                 search         CRC Error Frames       0
Lost Sync Count           0         RS Corrected Frames    0
Flywheels                 0         RS Uncorrectables      0
Lock Frames               0         Deleted Frames         0
Flywheels Output          0         Passed Frames          0
Slipped Frames            0         CADUs                  0
True Frames               0         Unrouteable CADUs      0
Inverted Frames           0         Fill CADUs             0

                    Terra MODIS                       2007/303/17:26:51
Loaded    Go: 2007/303/14:59:06   Stop: 2007/303/15:03:32      No Data
```

**RT-STPS Viewer**

Click on the "load" button, and choose "terracotta.xml" from the "config" subdirectory. This loads the Server with a Terra setup file. It will capture MODIS packets and store them in an Earth Observing System (EOS) Production Data Set (PDS) file in the data subdirectory. Press the "go" button, and the RT-STPS Server is ready.

**NOTE:** If you have a firewall, then you may not be able to run the Viewer, and you may have difficulties with other tools, unless you turn off, or make accommodations to go through, the firewall. The Server accepts data by default on port 4935. The output port numbers, if any, are defined in the setup files. The viewer talks to the Server using the Java RMI protocol. It initially connects through port 1099, but afterwards they talk through anonymous ports, so you may not be able to get the Viewer to work through a firewall.

### RT-STPS Viewer Function Summary

The RT-STPS Viewer functions are used to configure the RT-STPS Server and examine its status.

## Menu Bar

The Menu Bar contains the File, Commands, and Status pull-down menus.

### File Menu

The File Menu contains the program Exit item.

### Commands Menu

The Commands Menu contains the following commands to configure and run the RT-STPS Server:

a) Local Load. Displays a Dialog Box to select and load a configuration file stored on the computer executing the Viewer.

b) Remote Load. Displays a Dialog Box to select and load a configuration file stored on the computer executing the RT-STPS Server.

c) Go. Starts RT-STPS Server data processing.

d) Stop. Halts RT-STPS Server data processing.

e) Unload. Removes the current configuration from the RT-STPS Server.

f) Zero Status. Resets the statistics display.

### Status Menu

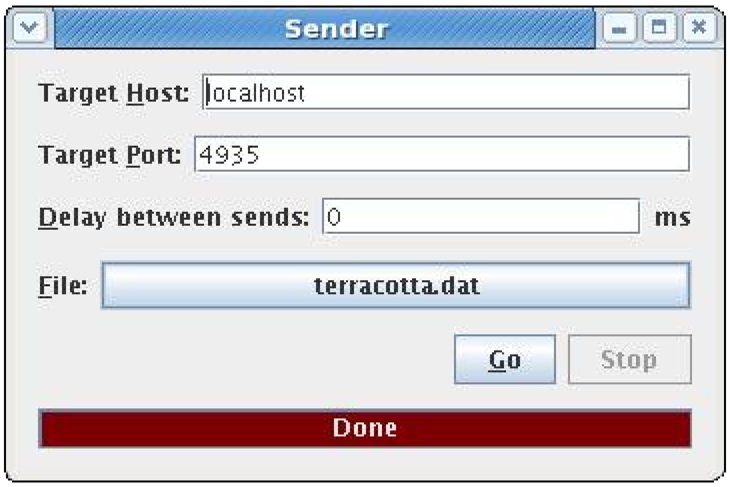
The Status Menu contains menu items to display Path Service Status, Packet Status, the Virtual Channel Status Table, and the Packet Status Table.

## Button Bar

Contains buttons linked to the Commands Menu Items (see above).

## RT-STPS Sender

This software package also includes the RT-STPS Sender, depicted below, which can be used to send a raw data file to the RT-STPS Server.



Target Host: localhost
Target Port: 4935
Delay between sends: 0 ms
File: terracotta.dat

**RT-STPS Sender**

Run:

Linux: ./bin/sender.sh
Windows: .\bin\sender.bat

The host should be "localhost", and the port number should be 4935. Leave the delay as zero. Click on the "File" box. Select the "data" subdirectory, then select the "terracotta.dat" file, and press "go" to send. You can watch the Server in the Viewer window. When the Sender finishes, the Server should automatically shut itself down. (Sender.sh simply sends file data to a port and host. It is not specifically written for the RT-STPS Server, so you might find it useful in other applications.) The output PDS files will appear in the data subdirectory.

**Rat**

Rat is a rate-buffering program that can spool data to a slow target. It may be useful when you load a configuration into the Server that sends data units to a remote target directly through a TCP/IP socket. A slow target will slow down the server too, and it may cause overruns in a real-time environment. You can run Rat on the same computer as the Server, or you can run it at a remote location. Rat.sh requires three arguments when it is invoked:

Linux: ./bin/rat.sh <inputPort> <targetHost> <targetPort>
Windows: .\bin\rat.sh <inputPort> <targetHost> <targetPort>

For additional information on Rat, refer to Appendix C.

# Appendix A
# RT-STPS XML Configuration Files

An RT-STPS configuration file defines the data pathways through the RT-STPS. (Configuration file examples are described in Appendix B.) The RT-STPS is a collection of connected nodes, and the configuration file defines each node's setup and how the nodes are linked together. Nodes pass data among themselves as either packets, frames, or units. A packet is a CCSDS Version 1 packet. A frame is a higher level container with a synchronization pattern and often Cyclic Redundancy Check (CRC) or Reed Solomon parity attached to the end. A CCSDS Channel Access Data Unit (CADU) is a type of frame. A unit is anything else including CCSDS Virtual Channel Data Units (VCDUs) and bitstream Bridge Protocol Data Units (BPDUs). Each node will generally accept at most one type of data unit, and most nodes will send one type of data unit.

An RT-STPS configuration file is XML-compliant, and you can see the general layout in the Document Type Definition (DTD) file, "rt-stps.dtd". Each element usually corresponds to one node with one exception. The standard Pseudo-noise (PN) decoder node is embedded in the frame synchronization element. The file format is complex, so it is best if you copy an existing file and change it to suit your needs.

Each RT-STPS configuration must begin with a frame synchronization node, but almost every node after that is optional. Most configurations will contain more than one internal stream. A typical full configuration path is as follows:

- The Frame Synchronizer receives blocks of bits. It finds frames and sends them to linked nodes. This is the only node that does not accept frames, packets, or units. It also attaches a time tag and quality information to each frame, to which subsequent frame handlers contribute.

- The PN Decoder receives frames and removes pseudo-noise from them. It sends the new frames to linked nodes.

- The CRC Decoder checks for CRC errors in each frame. It can discard bad frames. It sends frames to the next nodes.

- The Reed Solomon Decoder check for block RS errors in each frame. It can attempt to correct errors, and it can discard bad frames. It sends frames to the next nodes.

- The Frame Status node collects status information about frames, which it accumulates in counters. It is particularly useful if you are using the RT-STPS viewer to monitor session activity. Without it, most of the frame counters would remain zero. It gets most of its information from the frame quality. This node forwards the frames without modification.

- The Terra Decoder is a non-CCSDS-compliant PN decoder especially designed to handle a Terra spacecraft requirement. It accepts only Terra CADU frames and sends frames (CADUs).

- The CADU Service node accepts only CADU frames. It splits the single input stream into one or more independent streams based upon the spacecraft ID and Virtual Channel ID (VCID) that it finds in each CADU. You must configure this node so that spacecraft IDs and virtual channel IDs map to other nodes. It will discard any frame that has an unrecognizable mapping. You may link one "spid/vcid" combination to more than one node, or you may map more than one "spid/vcid" to a single node. Usually you map to CCSDS service nodes, but you could map a stream directly to an output device node.

- The Path Service node accepts CADU frames from one virtual channel, which it deconstructs and assembles into CCSDS packets. Usually you link one or more Path Service nodes to the CADU Service node. The Path Service node splits its input stream into one or more packet output streams based upon the packet's application ID. You must configure this node so that the packet application ID maps to other nodes. It will discard any packet that has an unrecognizable mapping. As with the CADU Service node, you may map multiple "appids" to one node or one "appid" to multiple output nodes. Usually you map the Path Service node to Packet nodes, but you could map it directly to output device nodes.

- The VCDU node is another node that you typically link to the CADU Service node. It accepts frames/CADUs. It removes header and trailer information and sends the resulting VCDUs or CVCDUs to one or more listening nodes. The receiving nodes must accept units.

- The Bitstream node is another node that you typically link to the CADU Service node. It accepts frames/CADUs. It extracts BPDUs and sends them to listening nodes, which must accept units.

- The Packets element is not a node itself but a container that lists Packet nodes. Each Packet node accepts CCSDS packets. Since each node performs packet sequence checking, it expects the packets to come from one packet stream, which is usually one application ID. It sends packets to packet listener nodes.

There are several kinds of output device nodes, which send packets, frames, or units to either files or TCP/IP sockets. Each output device node will usually only accept data units of a particular type. You may attach an output device node of the correct type to any node that listed above. Output device nodes do not send data units to other nodes, so you cannot link them as you do other nodes. The "output_channels" element defines all output device nodes in a configuration file.

The file, socket, and null output channels send annotated or non-annotated data units to the device.  The null device is a discard channel.  Annotation is quality and time, and you may optionally append or prefix it to each data unit.  The "sorcerer" output channel creates EOS Terra or Aqua spacecraft Production Data Set (PDS) and Expedited Data Set (EDS) files.  PDS and EOS Data Sets each include a Construction Record and one or more packet files containing un-annotated packets.  If you choose an annotation option, then the RT-STPS writes 64 bits of frame annotation with each frame, packet, or unit.  In the packet case, it also writes 32 bits of packet annotation before the frame annotation.

| **Bits** | **Packet Annotation (32 bits) Precedes Frame Annotation** |
|---|---|
| 31-18 | Not used (most significant bit) |
| 17 | 1= packet has invalid length, which is outside the configured minimum and maximum packet length for this packet stream. |
| 16 | 1= this packet could not be constructed in its entirety, and so it has appended fill data. |
| 15-0 | The number of "good" bytes in this packet.  For complete packets, it is the packet length.  For packets with fill, it is the index of the first fill byte. |

| **Bits** | **Frame Annotation (2 x 32 bits)** |
|---|---|
| 31-26 | Not used (most significant bit) |
| 25 | 1= Frame contains an idle/fill VCDU (CCSDS state). |
| 24 | 1= Frame has bad first header pointer (CCSDS error). |
| 23 | 1= Path Service had problem composing a packet from this frame. |
| 22 | 1= sequence error between this frame and preceding frame |
| 21 | 1= Frame is Reed Solomon uncorrectable. |
| 20 | 1= Frame is Reed Solomon corrected. |
| 19 | 1= Frame has CRC error. |
| 18 | 1= Slipped frame (Frame was aligned.) |
| 17 | 1= Inverted frame (Polarity was corrected.) |
| 16 | 1= Lock frame |
| 15-0 | Day of year (1-366) |
| 31-0 | Milliseconds of day |

The contents of every RT-STPS configuration file must be contained between the statements *<rt_stps id="title">* and *</rt_stps>*.  You should substitute a short description for the *title* field.  The RT-STPS Viewer will display it on the status line when it loads the configuration.

**The Frame Synchronizer Node**

The element name and link name is "frame_sync". Exactly one "frame_sync" node must appear in every configuration. This node creates frames from unsynchronized bits.

| Field | Default | Description |
|---|---|---|
| pattern | 0x1ACFFC1D | The frame synchronization pattern. The default is the CCSDS standard, and you should not change it unless you must process non-CCSDS telemetry. The number of characters in the pattern determines the pattern length. It must be at least two bytes long. |
| frameLength | 1024 | The frame length in bytes. If the frames contain Reed Solomon parity, then you must specify a frame length that satisfies the Reed Solomon decoder. 1024 is the required frame length for RS interleave 4 frames. If you have interleave 5 frames, use 1264. Interleave 1 frames are 256 bytes. |
| slip | 0 | If 1 or 2 and if the Frame Synchronizer does not see the pattern at its current anchor position, it will look forwards and backwards 1 or 2 bits to see if the frame has "slipped." If it finds sync, it will reset its anchor. This field is only meaningful if the Frame Synchronizer is in lock and it suddenly drops sync because of a slip. Setting this field should have little effect on performance, and there is no harm in setting it. In practice, frames are rarely slipped. The field must have a value of 0, 1, or 2. |
| trueSync | true | If true, the Frame Synchronizer searches for the pattern exactly as specified. There is no reason to set this field to false unless you are certain that the frames will have inverted sync, in which case you will save some processing time. If false, then you must set invertedSync to true. |
| invertedSync | false | If true, the Frame Synchronizer will search for an inverted sync pattern. When it detects an inverted pattern, it assumes that the entire frame is inverted. If false, then you must set trueSync to true. |

| Field | Default | Description |
|---|---|---|
| correctPolarity | true | If true, the Frame Synchronizer will correct the polarity of frames that it determines have an inverted sync pattern. It inverts the entire frame and not just the pattern. This field is meaningless unless invertedSync is true. This field must be true if it detects inverted frames and you subsequently route them to the Reed Solomon decoder or to any CCSDS processing nodes. The only time you might do otherwise is if you send frames directly from the Frame Synchronizer to an output channel. |
| flywheelDuration | 0 | This field affects how the Frame Synchronizer behaves if it drops sync after lock. If non-zero, it will skip over this many blocks of "length" bytes before it again searches for the pattern. The skipped blocks are called flywheel frames. Leave this field as zero for most applications unless you have some reason to skip chunks of data after a loss of synchronization. |
| sendFlywheels | false | If true, the Frame Synchronizer will send on flywheel frames as if they were lock frames. Usually it discards them. |
| PnEncoded | false | If true, the Frame Synchronizer assumes the frames are encoded with bit transition density encoding, and it will decode the frames. (This is also known as PN randomization.) The PN decoder is actually a separate RT-STPS node (link name "pn"), which is embedded in the Frame Synchronizer setup because it does not have any additional setup fields. |
| epoch | 19950810000000 (Aug 10, 1995 00:00:00) | The epoch, sessionStart, and stepSize fields configure the clock that the RT-STPS uses to create frame annotation. This may be important to you if you are writing annotation with each packet, frame, or unit, or if you are creating EOS PDS files through the "sorcerer" node. The epoch sets the start time from which all times are measured. Its format is a string of the form: "YYYYMMDDhhmmss." |

| Field | Default | Description |
|---|---|---|
| sessionStart | The current wall clock time. | This will be the time of the first data unit. By default, RT-STPS sets the session start time to the computer’s current time. By changing session start, you can have the session appear to run at a different date and time. The format for specifying sessionStart is the same as epoch. Omit this field to get the default behavior. |
| stepSize | 0 | Normally, the time difference between frames will be real time, after adjusting for epoch and the session start time. If you set this field to a positive value, then the annotation timestamps of successive frames will differ by this step size (in milliseconds), and the wall clock is ignored. For example, if you set stepSize to 100, then each frame's time will differ from the preceding one by 100 milliseconds. Warning: The frame synchronizer will not adjust the time to account for sync dropouts, so do not rely on the step size to detect lost frames. It will adjust for flywheel frames, dropped or not, however. |

### The CRC Decoder Node

The element name and link label is "crc". Only one CRC Decoder node may appear in a configuration. This node checks frames for CRC errors and may discard frames if it detects errors. The frames must have 16-bit CRC parity. Otherwise, either omit this node's setup, or bypass it in the links setup. Usually you will not change CRC fields.

| Field | Default | Description |
|---|---|---|
| includeSyncPattern | false | When true, the CRC Decoder will include the synchronization pattern in the CRC calculation. This is an atypical setup. |
| discardBadFrames | true | After the decoder calculates CRC and compares it against the parity in the frame, it can either discard the frame or pass it on to the next node. If it passes it on, a field in the quality annotation will mark it as a frame with a CRC error. In either case, the CRC error will appear as a count in the RT-STPS Viewer's display. If you pass frames with CRC errors, be aware that subsequent nodes may also encounter non-fatal processing errors depending on where in the frame the error occurred. In addition data units may be routed to incorrect destinations, or data units may have incorrect science or engineering data, because of the indeterminate bit errors in the frame. |
| offsetToParity | 0 | This is the byte offset from the frame start to the first byte of CRC parity, which is two bytes wide. In general, the CRC parity follows the frame data but precedes any Reed Solomon parity. If set this field to zero, then the RT-STPS calculates the value. Change it only if you have a non-standard location for the parity. |
| startSeed | 0xFFFF | This is the start value for calculating the CRC parity. It is usually all ones or all zeroes. If you encounter CRC errors on every frame and are certain that the frames do contain CRC parity, try setting this field to zero. |

**The Reed Solomon Decoder Node**

The element name and the link name is "reed_Solomon." Only one Reed Solomon Decoder node may appear in a configuration. This node checks frames for block Reed Solomon errors, and it can attempt to correct the errors if so configured. It may discard frames if it detects errors. The frames must have Reed Solomon parity. Otherwise, either omit this node's setup, or bypass it in the links setup.

This node does not perform CCSDS VCDU header error detection and correction.

If Reed Solomon parity is present, then the length of frames is preset to certain absolute values. For interleaves 1 through 5, the standard CCSDS frame lengths are 256, 512, 760, 1024, and 1264 respectively.

In typical RT-STPS processing, over 90% of the CPU time is spent doing Reed Solomon detection and correction. If you are having performance problems in a real-time environment, try turning off block correction.

| Field | Default | Description |
|---|---|---|
| interleave | 4 | This field's value must match your spacecraft's Reed Solomon interleave value. Only values between 1 and 5 inclusive are allowed. |
| doBlockCorrection | true | If true, the RS Decoder will attempt to correct any frame errors. It will then mark the frame's quality annotation as corrected if it corrected the frame, or as uncorrectable if it could not fix the errors or if this field was set to false. |
| discardUncorrectables | true | If true, the RS Decoder will discard any frame that it cannot correct, or any frame that has an error and doBlockCorrection was turned off. As with passing on frames with CRC errors, passing on frames with RS errors will almost certainly cause errors in subsequent nodes as well as incorrect routing or corrupted science or engineering data. |
| useStandardCCSDS | true | If true, the RS Decoder will be configured to use a standard CCSDS setup, which includes the virtualFill and dual fields as well as several hidden fields. You should never set this to false unless you have very good reasons. |
| virtualFill | -- | The number of virtual fill bytes in each frame. |
| dual | true | Dual mode or non-dual mode. |

## Spacecrafts

The element name is "spacecrafts". Only one spacecrafts element may appear in a configuration. This setup is not a node, but instead it is a list of spacecraft, and it is only required when you plan to do CCSDS processing. The spacecrafts element contains one or more spacecraft elements. Each spacecraft element defines information about a different spacecraft. The table describes the fields in a spacecraft element; the spacecrafts element has no fields.

| Field | Default | Description |
|---|---|---|
| label | None | The label is a unique name for this element, and other nodes will refer to this element by the label. There is no default. You must provide a label that is unique to entire configuration. Case is significant. Usually you will use the spacecraft name. |
| id | None | This is the spacecraft ID, a number. You must provide this field. There is no default. |
| insertZoneLength | 0 | Some CCSDS setups will have an Insert Zone embedded in every CADU. You must provide the zone length in bytes even if you do not plan to process it. The default is zero bytes, no Insert Zone present. |
| headerErrorControlPresent | false | If true, then each CADU VCDU header has special parity included in the CADU. If the error control field is present, you must set this field to true even if you do not plan to detect or correct VCDU header errors. |
| doHeaderDecode | false | This option is not currently implemented. If true, the RT-STPS would detect and correct errors in the CADU VCDU header. Header error control information and parity must be present if this field is true. |

## The Terra Decoder Node

The element name is "terra_decoder". The Terra Decoder is a special PN decoder for the Terra spacecraft that decodes a special non-standard PN encoding. Omit it for any other spacecraft. The link name is its label. It has no arguments other than a required unique label field. The usual way to specify it is:
<terra_decoder   label="TerraDecoder" />.

## The CADU Service Node

The element name and link label is "cadu_service". Only one CADU Service node may appear in a configuration. It is the entry node for all CCSDS processing. The node accepts frames and interprets them as CCSDS CADUs. It then routes the frames based upon virtual channel and spacecraft numbers to different CCSDS service nodes. The CADU Service node has no configuration for itself, but it does contain a list of one or more mappings of spacecraft ID and virtual channel ID pairs to CCSDS service node labels. This is one of two nodes where links are not defined under the links element. The element name for a member of its map list is "svlink," and the following table describes "svlink" fields. Note that multiple (spid, vcid) pairs may map to the same target, and one (spid, vcid) pair may map to multiple targets. CADUs with unmapped (spid, vcid) pairs are counted and discarded. Typically, you will map pairs to elements in the "ccsds_services" list, but this is not required. For example, you could map an "svlink" to an output channel.

| **Field** | **Default** | **Description** |
|---|---|---|
| spid | None | A spacecraft ID, a number, which will be found inside a CADU. You must provide this field. |
| vcid | None | A virtual channel ID, a number, which will be found inside a CADU. You must provide this field. |
| label | None | The label of a target RT-STPS node that expects CADUs or frames. All CADUs with matching spid and vcid will be sent to this node. You must provide this field. |

## The CCSDS Services Element

The element name is "ccsds_services". This is not an RT-STPS node, but instead it is a list of CCSDS service nodes. There are three different elements that may be in the list: vcdu, bitstream, and path. However, there may be zero or more elements of each type in the list. They may be arranged in any order. Typically, you will map the "cadu_service" node to these service nodes. The next three sections define the settings for each of the three service nodes.

## The VCDU Service Node

The element name is "vcdu". There may be more than one VCDU node. Each one must have a unique label. The VCDU node removes the VCDU from a CADU and sends it on to a node that accepts units. It can also send CVCDUs, which are VCDUs with Reed Solomon parity still attached. One VCDU node usually processes VCDUs for one virtual channel.

| **Field** | **Default** | **Description** |
|---|---|---|
| label | None | The label uniquely identifies this node, and it must be different from any other node label in the configuration. The name typically incorporates the virtual channel number. You must provide a label. |
| spacecraft | None | A reference to a spacecraft label, which you defined in the spacecrafts list. There is no default; you must provide a spacecraft label. |
| discardRsParity | false | If true, the VCDU node will discard Reed Solomon parity from each VCDU before forwarding it. The node gets the parity length from the Reed Solomon node, so you must configure the Reed Solomon decoder node to use this option. The Reed Solomon element must be in the configuration file, but it need not be a linked element. |

## The Bitstream Service Node

The element name is "bitstream". There may be more than one Bitstream node. Each one must have a unique label. The Bitstream node removes the BPDU from a CADU and sends it on to a node that accepts units. One Bitstream node usually processes BPDUs for one virtual channel. It does not merge BPDUs.

This node accounts for Reed Solomon and CRC parity by searching for the existence of those decoder nodes, even if they are not linked into the pipeline. If your CADUs do not have Reed Solomon parity, make sure you remove the Reed Solomon element from your setup. Merely bypassing it is insufficient. Otherwise, your BPDUs will be truncated. The same is true for CRC parity; remove the element definition if CRC parity is not present.

| **Field** | **Default** | **Description** |
|---|---|---|
| label | None | The label uniquely identifies this node, and it must be different from any other node label in the configuration. The name typically incorporates the virtual channel number. You must provide a label. |
| spacecraft | None | A reference to a spacecraft label, which you defined in the spacecrafts list. There is no default; you must provide a spacecraft label. |
| OCFpresent | false | If true, the node expects that every CADU will contain a 32-bit Operational Control Field (OCF). The OCF (also known as the Command Link Control Word or CLCW) is echo information from the forward command link. The Bitstream node uses this flag to compute the BPDU length. Incorrectly setting this field will cause BPDUs to be short or long by four bytes. |
| crcParityPresent | false | The Bitstream node uses this flag to compute the BPDU length. If true, it will subtract two bytes from the BPDU length. If false, it will look for the CRC Decoder node and flip this flag to true if it finds it, and then it will subtract two bytes. Therefore, if you leave this field as false, the node will automatically detect for CRC parity presence. Otherwise, if true, it will assume CRC parity is present regardless of whether or not the CRC Decoder node is available. |

## The Path Service Node

The element name is "path". There may be more than one Path node. Each one must have a unique label. The Path node performs packet reassembly on a CADU's data zone and sends the packets on to a node that accepts packets. One Path node usually processes CADUs from one virtual channel.

This node accounts for Reed Solomon and CRC parity by searching for the existence of those decoder nodes, even if they are not linked into the pipeline. If your CADUs do not have Reed Solomon parity, make sure you remove the Reed Solomon element from your setup. Merely bypassing it is insufficient. Otherwise, your packets may be truncated, and the node will report numerous dropouts and sequence errors. The same is true for CRC parity; remove the element definition if CRC parity is not present.

The Path Service Node sorts packets by their application IDs, and it can send them to more than one packet processing node based on the sorting. The node contains a list of one or more mappings of application IDs to packet processing node labels. This is one of two nodes where links are not defined under the links element. The element name for a member of its map list is "pklink," and the table that describes "pklink" fields follows the Path Service Node table. Note that more than one application ID may map to the same target, and one application ID may map to multiple targets. Packets with unmapped application IDs are counted and discarded. Typically, you will map application IDs to elements in the "packets" list, but this is not required. For example, you could map a "pklink" to a packet output channel.

| **Field** | **Default** | **Description** |
|---|---|---|
| Label | None | The label uniquely identifies this node, and it must be different from any other node label in the setup. The name typically incorporates the virtual channel number. You must provide a label. |
| Spacecraft | None | A reference to a spacecraft label, which you defined in the spacecrafts list. There is no default; you must provide a spacecraft label. |

| Field | Default | Description |
|---|---|---|
| maxRationalPacketSize | 8192 | This is the maximum rational packet size in bytes. If a packet size exceeds this value, the Path Service assumes that either the data are not really packet data or it is hopelessly lost in the current frame. If the packet length fails this test, then the Path Service stops any further processing for the current frame and resets itself to start looking for a new packet in the next frame. It discards the failed packet. Normally if you discarded CADUs with bad parity, then packets with bad lengths should rarely happen. Of course, make sure this field is larger than your packet size. |
| Fill | 0xC9 | When the Path Service is unable to fill a packet in its entirety, it will fill the remainder by repeatedly appending this fill byte. (However, it will discard any packet that does not have a packet header and at least one byte of real data.) It marks the packet annotation for packets with fill data. |
| OCFpresent | false | If true, the node expects that every CADU will contain a 32-bit Operational Control Field (OCF). The OCF (also known as the Command Link Control Word or CLCW) is echo information from the forward command link. The Path node uses this flag to compute the data zone length. Incorrectly setting this field will cause short packets and sequence errors because it will include OCF data in the data zone. |

| Field | Default | Description |
|---|---|---|
| crcParityPresent | false | The Path node uses this flag to compute the data zone length. If true, it will subtract two bytes from the data zone length. If false, it will look for the CRC Decoder node and flip this flag to true if it finds it, and then it will subtract two bytes. Therefore, if you leave this field as false, the node will automatically check for CRC parity presence. Otherwise, if true, it will assume CRC parity is present regardless of whether or not the CRC Decoder node is available. |
| discardPacketsWithFill | false | If true, the node will discard short packets to which it added fill data. If true, it will send them on, but it will also mark them in the packet annotation. |
| discardIdlePackets | true | If true, the node will discard idle packets. Idle packets are fill packets that usually contain no useful data. You usually want to discard idle packets unless there is other information in them that you need, such as in the secondary header. If the node does send them on, it marks them as idle in the packet annotation. |

This is the "pklink" element. One or more of these elements may be in the Path Service "packet" list. Each one maps an application ID to a label, which corresponds to a target node that accepts packets.

| Field | Default | Description |
|---|---|---|
| appid | None | The application ID. You must provide a value. |
| label | None | The label of a node that accepts packets. You must provide a label. |

### The Packets Element

The "packets" element is not a node but is a container for a list of one or more "packet" elements.

There may be more than one Packet node. Each one must have a unique label. The Packet node usually handles packets for one application ID. Its primary functions are to verify that each packet has an acceptable size and to look for sequence errors, which indicate a loss of one or more packets.

| Field | Default | Description |
|---|---|---|
| label | None | The label uniquely identifies this node, and it must be different from any other node label in the setup. The name typically incorporates the application ID. You must provide a label. |
| appid | None | An application ID associated with this node. This field is optional and has no effect on processing. It is here simply for documentation. |
| minSize | 15 | The minimum packet size that this node accepts. If it detects a packet with a length that is not within minSize and maxSize inclusive, it will mark the packet's annotation, and it may delete the packet if so configured. |
| maxSize | 8192 | The maximum packet size that this node accepts. If you want this node to only pass packets of one size, set maxSize and minSize to the same value. |
| discardWrongLengthPackets | true | If true, the node discards packets with sizes that are not within minSize and maxSize. Otherwise, it marks the packet annotation and sends them on to the next node. |
| checkSequenceCounter | true | If true, the node checks the packet sequence counters for packet gaps. It reports the number of gaps and cumulative count of missing packets in its status. It also marks packet annotation for packets that are near gaps. |

## Output Channel Nodes

Output Channel nodes write data units to output devices. They do not send data units to other nodes. Currently you may configure output to TCP/IP sockets, files, and null devices. Files may be RT-STPS custom format files or the "Sorcerer" node, which is a special node that creates EOS PDS and EDS file sets.

The "output_channels" element is a container for a list of output channel nodes. You may have any number of output channel nodes in a configuration. One output node may handle data units from multiple node sources.

Most output channel nodes will accept only one data unit type: packets, frames, or units.

The "null" channel simply discards data. It can accept any data type. Its only field is its label, which must be unique.

### The RT-STPS Output Channel Node

The RT-STPS format output node can be configured to handle packets, frames, or units, but not more than one type simultaneously. This node can then be wired to a socket or a file. Finally, you can configure it to write non-annotated data units, data units with appended annotation, or data units with annotation preceding each data unit. The element name defines the device. Use "file" to write data units to a file. Use "socket" to write data units to a TCP/IP socket.

**The File Output Channel Node**

The element name is "file," and there may be more than one element.

| **Field** | **Default** | **Description** |
|---|---|---|
| Label | None | The label uniquely identifies this node, and it must be different from any other node label in the setup. You must provide a label. |
| UnitType | None | UNIT, PACKET, or FRAME. This field describes the type of data unit this node will accept. Case is significant. You must provide a value. |
| annotation | None | NONE, BEFORE, or AFTER. This field determines if annotation is to be written with each data unit and where it should go. Frames and units get frame annotation. Packets get packet annotation followed by frame annotation. NONE means the data units are written without annotation. Use BEFORE to write the annotation before each data unit. Use AFTER to append the annotation. |
| Directory | -- | The output file directory. This field has no meaning for socket elements. If you omit this field, the node will create files in the default data directory, which is usually "data". The default directory is defined as an argument in the script that starts the server. If you choose to put your files in a different directory, then it is best to create a subdirectory of "data" and put them there. If you attempt to create files in a different directory tree, file creation will probably fail. The RT-STPS server has security restrictions about where users may create files. If you wish to write files elsewhere, you will need to edit the file rt-stps.policy to unlock your directory before you start the server. |
| Filename | -- | The name of the file that this node creates. If you set autoGenerateFilename to true, then omit this field. |
| UserLabel | -- | This is an optional label that the node inserts into the file name when you set autoGenerateFilename to true. |

| Field | Default | Description |
|---|---|---|
| autoGenerateFilename | true | If true, the node generates a unique filename for you that should not overwrite an existing RT-STPS file. The file name will have the form "t"+ "p or f or g" + "yyyyDDDHHmmss"+ "userLabel" + ".dat". The "t" stands for "telemetry." It then inserts "p", "f", or "g" for "packets", "frames", or "general" units. It finally adds the date and your user label. |

**The Socket Output Channel Node**

The element name is "socket," and there may be more than one element. To use sockets, there must be a client at the target computer waiting at a server socket to establish a connection. If there is no server, then trying to load a configuration with socket output will fail immediately.

A socket connection sending data units to a server can be a considerable bottleneck for the RT-STPS. The RT-STPS does not use threads to write to sockets, so a slow or sluggish target will affect performance and could even stop data processing. If this problem occurs, consider using the DRL rate buffering program. This is a separate program that lies between the RT-STPS and the target. If the target read rate is slow, the rate buffering program will spool data to a temporary disk file. See the tools section for more information.

| Field | Default | Description |
|---|---|---|
| Label | None | The label uniquely identifies this node, and it must be different from any other node label in the configuration. You must provide a label. |
| UnitType | None | UNIT, PACKET, or FRAME. This field describes the type of data unit accepted by this node. You must provide a value. |
| Annotation | None | NONE, BEFORE, or AFTER. This field determines if annotation is to be written with each data unit and where it should go. Frames and units get frame annotation. Packets get packet annotation followed by frame annotation. NONE means the data units are written without annotation. BEFORE means the annotation is written before each data unit. AFTER means annotation is appended. |
| BufferSize | 8192 | The node configures the output socket to use this buffer size. Use of this value depends on the Operating System (There is no indication if this value is used). If you are trying to improve your network performance, experiment with this number. |

| Field | Default | Description |
|---|---|---|
| Host | None | The host name or IP address of the target computer. You must provide this field. |
| Port | None | The port number of the target computer. The RT-STPS will connect to this port. |

## The Sorcerer Node

The element name is "sorcerer," and there may be more than one element in a configuration. This output node creates an EDOS PDS or EDS file set. A file set consists of a Construction Record file, which is the equivalent of a shipping letter, and one or more data files. A data file contains packets without annotation for up to three application IDs. This node is more complicated than most because it contains sub-elements. You may wish to consult an EOS Interface Control Document to understand the significance of some of these fields.

| **Field** | **Default** | **Description** |
|---|---|---|
| Label | None | The label uniquely identifies this node, and it must be different from any other node label in the setup. You must provide a label. |
| major | 0 | The major version number. This value is inserted into the construction record and has no effect on processing. |
| minor | 0 | The minor version number. This value is inserted into the construction record and has no effect on processing. |
| spid | 42 | The spacecraft ID. This value is inserted into the construction record and has no effect on processing. |
| path | -- | The output file Directory. If you omit this field, the node will create files in the default data directory, which is usually "data". The default directory is defined as an argument in the script that starts the server. If you choose to put your files in a different directory, then it is best if you create a subdirectory of "data" and put them there. If you attempt to create files in a different directory tree, file creation will probably fail. The RT-STPS server has security restrictions about where users may create files. If you wish to write files elsewhere, you will need to edit the file rt-stps.policy to unlock your directory before you start the server. |
| datasetCounter | 0 | A number that is embedded in the construction record and the file name. |

| Field | Default | Description |
|---|---|---|
| create | -- | The creation date for the output files, which is inserted into the construction record. If you omit this field, Sorcerer uses the current date and time. If you provide it, Sorcerer expects this format: "yyDDDHHmmss". |
| KBperFile | 0 | Kilobytes per file. If you set this to zero, Sorcerer will create a construction record file and a data file that contains all packets from the session. If you set this field to a positive number, Sorcerer will split the data file into a set of data files, each one approximately the size you specify. Each file name will have a sequence number embedded in it. Note that the file name field has room for only 99 data files. If you set KbperFile so small that Sorcerer tries to create more, you will get unexpected and unpleasant results. |
| test | false | A flag embedded in the construction record to indicate that the PDS contains test data instead of real data. |
| type | PDS | This may be "PDS" or "EDS" file. It affects how Sorcerer names the output files. It does not affect processing except when QuicklookEDS is set to "true". |
| QuicklookEDS | false | This field is ignored unless the type is "EDS". When true and type is "EDS," Sorcerer only writes packets that have the quicklook flag enabled in their secondary header. Do not turn this flag on for non-Terra application IDs that do not use the Terra secondary header format. |
| discardBadLengthPackets | True | If true, Sorcerer will discard packets with an incorrect length. Otherwise, it will write them. |

### The Application ID Sub-Element

The element name is "appid". Each "appid" sets up one application ID. There must be at least one "appid" element, but there may be no more than three. Each "appid" may also contain a list of valid packet lengths for that application ID. You can either specify a range of valid lengths, or you can list the lengths individually.

| Field | Default | Description |
|---|---|---|
| id | None | Specifies the application ID number. This field is required. |
| vcid | None | The virtual channel number from which this application ID came. An application ID may come from a maximum of two virtual channels. You specify the second one in vcid2. This field is required. |
| vcid2 | None | A second virtual channel number. Omit this field if there is no second virtual channel for this application ID. |
| spid | 42 | The spacecraft ID. Use 154 for Aqua. |
| CUCtime | false | This field determines the expected packet secondary header format. If you do not set it correctly, then the time fields in the construction record will have an incorrect format. Set this to false for all Terra application IDs. For Aqua, you must consult the Interface Control Document. Some applications IDs use a nine byte secondary header (all Terra) in which the time field is in CCSDS Day Segmented format. |
| Stepsize | 1 | The packet sequence number step size. You should not change this number. Sorcerer uses it to determine the gap size for missing packets. |
| minLength | -- | The minimum packet length. It must be at least 15 and not bigger than maxLength. There are two ways you can configure packet lengths. Either set minLength and maxLength, or provide a list of "packetLength" elements, one for each length. If you provide a list, then Sorcerer ignores minLength and maxLength. However, you must use one of the methods to configure packet lengths. |
| maxLength | -- | The maximum packet length. |

### The Packet Length Sub-Element

The element name is "packetLength". Each statement defines a valid packet length for an application ID in this PDS. Some application IDs may have more than one packet length; you list them here, one per statement.

| Field | Default | Description |
|---|---|---|
| length | None | A packet length. When you supply a "packetLength," you must provide a length value. |

**Links**

The "links" element is not a node, but it is a list of data links between nodes. It defines the data flow path through the nodes. Each "link" element defines a "from" and "to" field. You must define a links path through your nodes. However, you must be careful to only link nodes that convey the proper data type. For example, if you link a node that sends frames to another node that expects packets, the configuration will fail.

| Field | Default | Description |
|---|---|---|
| from | None | The source node from which data units are sent. This is an element label and is required. |
| to | None | The destination node to which data units are sent. This is an element label and is required. |

The node name used in the link is the node's label. Some nodes are singletons, and they have predefined labels. Those are:

| Node | Label |
|---|---|
| Frame Synchronizer | frame_sync |
| PN Decoder | pn |
| CRC Decoder | crc |
| Reed Solomon Decoder | reed_solomon |
| Frame Status | frame_status |
| CADU Service | cadu_service |

The links list does not define all links. The ones in the list are unconditional paths. There is no filtering or sorting. The CADU Service node and the Path nodes also define links, but these links are conditional ones, subject to sorting based on virtual channel, spacecraft, and application ID. Unfortunately, you must deal with the links in all places. Within the links element, you usually must define the singletons down to CADU Service and then all links to output channels.

For your convenience, here is the standard links path from the frame synchronizer to CADU Service:

```
<link from="frame_sync" to="pn" />
<link from="pn" to="crc" />
<link from="crc" to="reed_solomon" />
<link from="reed_solomon" to="frame_status" />
<link from="frame_status" to="cadu_service" />
```

Modify this links path as needed. For example, if do not have CRC decoding, then remove the CRC lines and substitute: link from="pn" to="reed_Solomon." You then usually must add output channel links to the list as in this example:

```
<link from="vcdu18" to="file1" />
<link from="bitstream30" to="file2" />
<link from="a256" to="file3" />
```

# Appendix B
# RT-STPS Server Configuration

The RT-STPS Server is configured using command line arguments and system property definitions. You may pass one argument (the server name) when invoking the server. The full Server name then becomes "RtStpsServices." + name. The default full Server name is "RtStpsServices.A" if a name is not specified.

The server understands the following system properties. You set them when using the -D attribute (e.g. "-Dport=4935"). If not set, default values will be used as described.

a) -Dconfig=xmlConfigFileName. A configuration file to be used until overridden by a loaded file. The default name is "default.xml".

b) -Dport=4935. The port number that the Server reads for telemetry data. The default port is 4935.

c) -DbufferSizeKb=8. The amount of data to accumulate before processing. The default is 8 kb.

d) -Dsetup=configurationDirectory. The directory where local configuration files are located. If provided, all files must be within the directory tree. The default is "config".

e) -Dlog.stdout. If specified, log messages are written to the standard output.

f) -Dlog.file=<file>. If specified, log messages are written to <file>.

g) -Dlog.server=<host:port:tmpDir>. If specified, log messages are sent to the NSLS server at host:port (e.g., localhost:3500) and to the temporary directory tmpDir when the NISGS Status/Event Logging System (NSLS) server is unavailable.

If none of the "-Dlog.*" properties are specified, log messages are written to the standard output and a file named "rt-stps.log" by default.

These arguments and system properties are set in the "server.sh" (Linux) or "server.bat" (Windows) file, or in the configuration file for the JSW if the RT-STPS Server is being run using that method. Additional system properties may also be present but should not be modified.

## Automatic Setup

The Server automatically loads the last-loaded setup file if it unexpectedly receives data on port 4935 and it has yet to be configured. (You can test this by having sender resend terracotta.dat again to the Server.) If you expect to use only one setup, you can use this feature to have the Server run in an unattended mode. When the Server first starts, it uses "default.xml" as its default setup file, which does not exist. You can set up a different default by editing "./bin/server.sh" (Linux) or ".\bin\server.bat" (Windows) and changing "default.xml" to something else, or adding the "config" option. Remember though, that if you use the viewer to load a different setup, then the new setup becomes the new default. You should add (or edit) the following option to the server command line. It must appear before "gov.nasa.gsfc.drl.rtstps.server.TcpServer."

```
-Dconfig=yourSetupFileName
```

The Server will look for the file in its "config" subdirectory.

## Other Installation Issues

The Server has a name, and the default name is "A". You can change its name by appending it to the end of the command line in the "./bin/server.sh" (Linux) or ".\bin\server.bat" (Windows) file. The RT-STPS Server will handle only one input line; you must run a second server to handle a second channel. To run a second server, create a second "./bin/server.sh" (Linux) or ".\bin\server.bat" (Windows), edit it, and change the input port number to something other than 4935. Finally, give this server a different name, "B" for example.

The Viewer will only talk to server "A" by default. You must run a second viewer to talk to a second server. Copy and edit "./bin/viewer.sh" (Linux) or ".\bin\viewer.bat" (Windows) and add the name to the end of the command line. Notice by default that the Viewer talks to the Server on the local host. You can change "localhost" to a different name, and then you do not have to run the Viewer on the same computer as the Server. The Viewer first looks at the "configDir" parameter in the command line for the names of configuration files that it sends to the Server.

**NOTE:** Over 90% of the CPU time is used to do Reed Solomon detection and correction. If you are doing real-time data processing, you can bypass Reed Solomon correction to avoid overruns. If you configure the Server to write to a socket, note that a slow receiver will affect the Server and will cause overruns in a real-time scenario. A socket interface exists to the Server that will allow you to send text commands (load, go) to the Server. The other "sh" (Linux) or "bat" (Windows) files in the "bin" directory allow you to control the Server from the command line. For example, "./bin/load.sh" (Linux) or ".\bin\load.bat" (Windows) loads and enables the server.

## Setup

The "config" subdirectory contains sample setup files. An RT-STPS setup file is in XML, and you can see the template for all fields in the "rt-stps.dtd" file. These setup files are complex, so you may want to copy and change an existing setup file. If you examine a sample setup file such as "terracotta.xml" with a text editor, note that almost every element corresponds to an RT-STPS node. The RT-STPS is a collection of data linked processing nodes. [Note however, that the Pseudonoise (PN) decoder node is embedded in the Frame Sync (FS) setup.] Do not change the node order in the file.

The "links" element defines the data paths through the nodes. The first node is always the "frame_sync" node. The "links" paths are all unconditional branches. Some specific nodes ("ccsds_services" and "path") have conditional branches. For example, "ccsds_services" routes CADUs to target nodes based on spacecraft ID and virtual channel ID, and those links are found in the "ccsds_services" element.

Every node supports links to multiple nodes. For example, you could link the "frame_sync" node to "reed_Solomon" and to an output node, and the "frame_sync" node would send frames to both targets. This multi-linking also works for the "ccsds_services" and "path" special link definitions. If you examine "rt-stps.dtd", you may find additional fields with defaults that are not in the sample setup files. You may delete most element definitions from a setup file except for the "frame_sync" and "links" elements, provided that you do not link to or from them. The "terra_decoder" node handles the special, non-compliant PN encoding found in certain Terra virtual channels. The "reed_Solomon" node does not currently do CCSDS header correction and those arguments do nothing. Block detection and correction do work.

These sample XML files and others are in the "config" subdirectory:

a) default.xml. Default configuration loaded by RT-STPS if a configuration file is not otherwise specified.

b) aqua.xml. Reads Aqua Spacecraft Telemetry from socket 4935 and writes PDS and CSR data to files and a port for each of the major instruments aboard the spacecraft.

c) terra.xml. Reads Aqua Spacecraft Telemetry from socket 4935 and writes PDS and CSR metadata to files and a port for the MODIS instrument.

d) terracotta.xml. Used to read the terracotta.dat when testing the RT-STPS installation.

e) This configuration creates Spacecraft ID 42 VCID 42 MODIS APPID 64 packets, and is similar to terra.xml but does not send data to a port. See the "Program Operation" section.

f) framer.xml. Captures frame data from the Frame Synchronizer and writes these data directly to a file without any further processing. This configuration is used for testing and development.

# Appendix C
# RT-STPS Tools

## RT-STPS Command Line Tools

Scripting tools in the rt-stps/bin subdirectory to manage the RT-STPS programs are described below.

### Batch Script

The Batch Script starts the RT-STSPS Batch Processor to read a raw telemetry file and produce packet, frame, or other files as specified in the configuration file. (See the "Program Operation" section.) The format is:

```
Batch.sh  path/configurationFile  path/outputFile
```

### Server Script

The Server Script starts the RT-STPS Server. (See the "Program Operation" section.) The format is:

```
server.sh
```

### Viewer Script

The Viewer Script runs the RT-STPS Viewer. (See the "Program Operation" section.) The format is:

```
viewer.sh
```

### Sender Script

The Sender Script runs the graphical user interface to copy input files to a running RT-STPS Server for processing. The format is:

```
stop.sh  path/localConfigfileName
```

### GetStatus Script

The GetStatus Script retrieves status information from the RT-STPS Server on the local host. The format is:

```
getstatus.sh
```

### Load Script

The Load Script loads a local configuration file into the local RT-STPS Server. The format is:

```
load.sh  path/localConfigfileName
```

### Shutdown Script

The Shutdown Script halts data processing and unloads the current configuration file on the local RT-STPS. The format is:

Shutdown.sh

**Stop Script**
The Stop Script terminates the local RT-STPS.  The format is:

stop.sh

**Version Script**
The Version Script prints RT-STPS version information and exits.  The format is:

Version.sh

**Rat Script**
The Rat Script spools RT_STPS Server output to a low data-rate target.  The format is:

Rat.sh  inputPort  targetHostName outputPort.

## Sender

Sender is a graphic utility that sends files as TCP/IP packets to a designated host and port.  You can use it to send data files to the RT-STPS Server on port 4935, or you can use it to send any kind of data to any application.  It does not use any special protocols.

## Rat

Rat is a rate-buffering program that can spool data to a slow target.  It may be useful when you load a configuration into the Server that sends data units to a remote target directly through a TCP/IP socket.  A slow target will slow down the server too, and it may cause overruns in a real-time environment.  You can run Rat on the same computer as the Server, or you can run it at a remote location.  Rat.sh requires three arguments when it is invoked:

```
Linux:  ./bin/rat.sh <inputPort> <targetHost> <targetPort>
Windows:  .\bin\rat.sh <inputPort> <targetHost> <targetPort>
```

Rat acts as a server and listens for socket connections on its input port.  When a connection is made, it then in turn connects to the target on the target port.  If no one is listening, it closes down all the sockets and once again listens for connections on its input port.  It will only service one input connection at a time.  Once connected, Rat gets data from the input port and sends it to the output port.  It uses an internal memory queue to buffer data.  If the buffer fills, it then switches to a temporary file as a buffering device.  Even if the input connection closes, it will continue to send buffered data to the target.  When finished, Rat closes the output socket.  The target need not be alive and serving when you start Rat.  Rat only attempts to connect after the RT-STPS server has connected to it.  Once you start Rat, you should not have to restart it.

## Alternate Setup Interface

To configure the RT-STPS server, you either use the Viewer (or the command line load command, which uses the same interface as the Viewer), or you set up the Server so that it automatically loads the same setup file on each session. There is, however, an additional command interface to the Server, but you will need to do some programming to use it. This interface is useful if you want to integrate the RT-STPS into a larger system. For example, you would like to connect it to scheduling software.

The RT-STPS server listens for connections on port 5935. (The port number is fixed.) It expects to receive text string messages on this port. Each message should have the normal line terminator. Case is significant. The messages are commands to load and shut down sessions. The available commands are:

a) loadgo <configurationFileName>. The server will load the configuration file from its configuration directory. It then enables itself for processing. Example: "loadgo terracotta.xml".

b) shutdown. It stops processing and unloads the current configuration, which closes all output files.

c) quit (or a null string). This terminates your connection to the Server through port 5935. The Server will close any sockets and will wait for another connection attempt.

d) rloadgo <configuration>. You can use this form to send a complete configuration to the Server instead of using a setup that is local to the Server. The configuration must be one text string following "rloadgo" with no embedded line terminators. We have not tested this command, so use at your own risk.

The Server does not send responses to any of these commands. The only feedback you will get is via the Server's monitor window. It will print the usual load and shutdown messages. It will also print error messages labeled as "ProxyThread" messages if it encounters them.